Gameboy 模拟器 项目文档

项目信息介绍

项目名：Gameboy-Emulator-CPP
项目成员：赵屹雄（1750756）
项目 Github 地址：https://github.com/Chiro11/Gameboy-Emulator-CPP
项目进度时间线：
2019.5.8 实现 Z80 Opcode 指令集
2019.5.16 实现 Z80 CB Opcode 指令集
2019.5.18 实现 Memory 基础映射
2019.5.21 添加 ROM 载入部分
2019.5.26 实现 GPU Background 渲染
2019.5.28 整合指令集与 Memory 部分，创建游戏主循环
2019.6.2 添加 SDL 库支持
2019.6.3 整合 GPU 部分，调试，运行游戏封面成功
2019.6.4 添加对游戏的交互操作
2019.6.8 实现 GPU Sprites 渲染
2019.6.9 整合交互模块和完整 GPU 模块，运行井字棋成功
2019.6.11 实现 Memory Bank Controller，测试马里奥大陆，可正常加载
2019.6.13 添加 Timer 模块

项目开发文档

模拟器的显示部分基于 SDL 游戏库

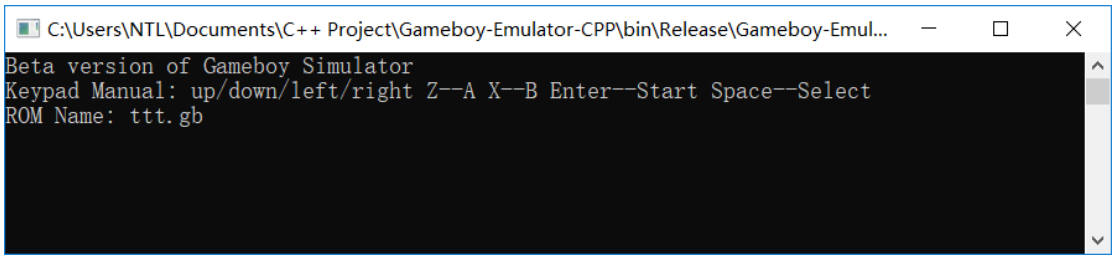
项目整体分为五个模块：CPU，Memory，GPU，Key 和 Timer

实现的基础功能：
- 实现 Gameboy Z80 CPU 模拟(支持大部分指令集)
- 实现时钟模拟
- 实现内存模拟
- 支持基本图形操作
- 支持对游戏的交互操作(即输入)
- 支持载入 ROM
- 可以基本玩一款 GB 游戏

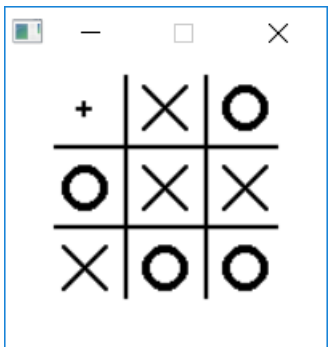
实现的可选功能：
- Background Window Graphic
- Sprite
- Memory Bank Controller

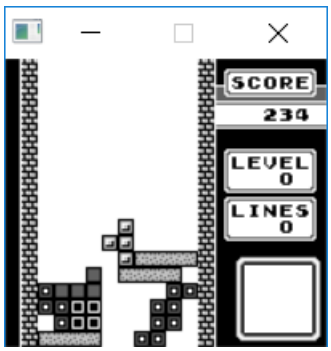
游戏运行：

将 ROM(即.gb 文件)与 exe 文件放入同一目录下，同时确保目录下包含 SDL2.dll，双击 exe 运行



输入 ROM 全名(如 ttt.gb)，回车运行，游戏测试画面如下





键位操作：

键盘上下左右分别对应手柄上下左右，Z 对应手柄 A 键，X 对应手柄 B 键，Enter(回车)对应手柄 Start 键，Space(空格)对应手柄 Select 键

程序逻辑：

模块初始化，载入 ROM，执行主循环

```
while(key.press()) {
    time = cpu.step();
    gpu.step(time);
    timer.step(time);
}
```

每一次循环中，检查键盘输入，CPU 执行一条指令或进行 Interrupt 处理，GPU 根据时钟判断当前显示的阶段，Timer 进行计时

实现思路：

CPU 模块：如果 Interrupt Flags 对应标志位为 1，则执行中断处理，将 Program Counter(以下简称 PC)跳转到中断处理程序位置，否则读取 PC 位置指令并执行，有效指令共 496 条，其中 opcode 244 条，CB opcode 252 条

Memory 模块：分为读取和写入，Memory 不同区域有不同功能，将对应区域作映射即可，由于 Memory Bank Controller(以下简称 MBC)的存在会将 Cartridge 和 External RAM 映射变得复杂，需要特殊处理

GPU 模块：时钟部分模拟电子显像管，考虑水平消隐和垂直消隐，每次水平消

隐前将一行画面渲染到 Buffer 中，垂直消隐前将 Buffer 中的画面推送到 SDL 窗口 Surface 上。渲染部分先后渲染 Background 和 Sprite，逻辑较为复杂

Key 模块：利用 SDL 库中的 Event 机制判断键盘上的键位操作，并推送给 Memory，代码量少但是细节较多

Timer 模块：单独计时模块的主要作用是提供定时的 Interrupt 处理以及随机数生成种子。共有两个计时器：Divider Clock 的计时速率恒定，且溢出后自动归零；Counter Clock 可以调节四种计时速率，且溢出后赋值为特定值，并标志 Interrupt

项目难点：
每一个模块难以单独测试，需要把整个框架基本搭好，CPU，GPU，Memory 三个基本模块全部实现的基础上才能运行

编写 CPU 指令集工作量繁重

GPU 的渲染部分细节很多，参考了很多资料后才着手开始编写

交互部分，由于模拟器不能完全模拟原 Z80 CPU 的运行速度，所以对按键的检测周期和正常机器有差异

遇到的困难与解决方案：
CPU 指令过多，一秒执行数百万条，Debug 时难以定位问题
解决方案：将每条指令执行后的寄存器的值输出到文本文件中，二分法定位

每一个模块都有很多成员变量(类似寄存器和 Tile 数据)需要维护，但是写入内存以后需要分别考虑更新，使类之间的关系变得复杂且代码易错
解决方案：模拟原始硬件处理方案，需要时直接从内存读取，处理结束后再写入回内存，使得大部分模块只需对 Memory 建立接口即可

Interrupt 机制，手册上对中断处理优先级的描述不清
解决方案：经测试，大部分游戏只需要垂直消隐中断即可正常运行，加入其它中断处理反而会出问题